Implementation of D* Path Planning Algorithm with NXT LEGO Mindstorms Kit for Navigation through Unknown Terrains

C. Saranya[1], K. Koteswara Rao[1], Manju Unnikrishnan[2], Dr. V. Brinda[3], Dr. V.R. Lalithambika[4] and M.V. Dhekane[5]

[1] Scientist/Engineer, Control Design Division, Vikram Sarabhai Space Centre,India,
saranya_c@vssc.gov.in, k_koteswararao@vssc.gov.in

[2] Section Head, Control Design Division,
Vikram Sarabhai Space Centre, India, u_manju@vssc.gov.in

[3] Division Head, Control Design Division,
Vikram Sarabhai Space Centre, India, v_brinda@vssc.gov.in

[4] Group Director, Control and Guidance Design Group,
Vikram Sarabhai Space Centre,India. vr_lalithambika@vssc.gov.in

[5] Deputy Director, Control Guidance and Simulation Entity,
Vikram Sarabhai Space Centre, India, mv_dhekane@vssc.gov.in

***Abstract*— Autonomous Robots use various Path Planning algorithms to navigate, to the target point. In the real world situation robot may not have a complete picture of the obstacles in its environment. The classical path planning algorithms such as A*, D* are cost based where the shortest path to the target is calculated based on the distance to be travelled. In order to provide real time shortest path solutions, cost computation has to be redone whenever new obstacles are identified. D* is a potential search algorithm, capable of planning shortest path in unknown, partially known and changing environments. This paper brings out the simulation of D* algorithm in C++ and the results for different test cases. It also elucidates the implementation of the algorithm with NXT LEGO Mindstorms kit using RobotC language and evaluation in real time scenario.**

***Index Terms*— Path planning, D*, Robot Navigation**

I. Introduction

Autonomous robots are used in wide variety of fields ranging from domestic to industrial and interplanetary applications. Path planning algorithms are indispensible in any autonomous robot irrespective of its mission. When the robot has to drive to some target point automatically, considering the known and unknown obstacles on its way, there has to be continuous monitoring of its path and re-planning if required, to reach the target point successfully.

Rovers used for planetary mission are hardly aware of its environment. Rovers will be commanded to reach to some interested spot for conducting specific experiments. With the subtle information on the surrounding obstacles, the Rover is expected to reach the interested spot rather the target point without colliding against any obstacles. As the Rover identifies new obstacles it should build the map of the adjacent region and also re-plan its path if required.

Grid based search algorithms like A* and D* are classical methods used for path planning. Classical algorithms are guaranteed to find the path provided sufficient information is available.

Apart from grid based planning techniques, potential field and Rapidly Exploring Random Tree are some of the other approaches to the path planning problem [1][2]. Potential Field method treats the Robot as point configuration which attracts towards the goal while repulses from obstacles [3]. But the robot may get trapped in local minima and fail to find the path. Rapidly Exploring Random Tree technique is probabilistic sampling based approach which incrementally builds a roadmap [4]. The drawback with this algorithm is that the path identified may not be always optimal.

This paper focuses on implementation of D* algorithm with LEGO Mindstorms kit. This paper is categorized in the following manner. Section II gives a description of D* algorithm and Section III describes the customization done to D* algorithm to suit real time implementation. Section IV gives the implementation of the algorithm in high level language C++, while Section V describes the NXT LEGO Mindstorms implementation using RobotC. Section VI briefs the results and Section VII presents the concluding remarks.

II. D* Algorithm

D* is a Path Planning algorithm which belongs to grid based search algorithm. It is dynamic version of A* algorithm. A* algorithm [5] is a simple path planning algorithm in which the cost functions computed are fixed, whereas in the D* algorithm cost functions are re-computed and updated as and when new obstacles are detected[6]. D* is efficient for unknown terrains as it involves the cost updation.

Several path planning algorithms based on D* are implemented in autonomous rovers. D* Lite, is a simplified version of D* involving heuristics which gives efficient results [7] [8] [9] [10]. But it increases the overhead by producing underconsistent states. Delayed D* is an approach to overcome it [11]. Ref [12] deals with the implementation of

DOI: 03.LSCS.2013.2.23

Focused D*, for Unmanned Combat Aerial vehicle. Field D*, another variant of D*, does not restrict the rover's orientation to be in steps of 45 degree [13]. This paper's focus is to bring out implementation of D* algorithm customized to real time scenario by eliminating the usage of data structures such as Open and Closed List.

The sequence of steps in planning path is elaborated below:

Step 1: Logical construction of Grid: The area under exploration has to be logically split into small squares, to make an 'n x n' grid. Each cell in the grid has a state to denote the presence or absence of obstacles. Thus if there is no obstacle then the cell's state is Walkable and if there is an obstacle its state is Un-Walkable.

Step 2: Planning: The Rover has been assumed to be aware of its starting cell and it has a map of grid with states of each cell in its memory. Initially all cells are considered to be Walkable.

For each cell X, in the grid, let c(Y,X) denote the cost of moving from Y to X and h(G,X) denote the sum of the cost involved in moving from X to G, G being the Goal point. Let k(X) denote the minimum of h(G,X).

BackPointers bk(X)=Y, denoting the cell with the minimum cost among the adjacent eight, are assigned to each cell.

Step 3: Moving and Sensing: BackPointers are followed unless there is a discrepancy in the computed cost.

Step 4: Cost Modification: If the lately computed cost $h_{new}(G,X)$ is greater than the previously computed value, it leads to a discrepancy. Then, the state of the cell which has an obstacle is modified to Unwalkable and CostUpdate is called for.

```
If h_new (G,X) > k(X)
        Obstacle detected
        CostUpdate
End
CostUpdate
      recompute  h(G,X)
      find k(X)
       update bk(X)
End CostUpdate
```

While re-computing h(G,X) all the adjacent cells are to be considered to choose the new BackPointer which gives the minimum cost.

```
For each neighbour Y of X:
 If h(G,Y) d" k(X) and h(G,X) > h(G,Y) + c(Y,X) then
        b(X)=Y
        h(G,X)=h(G,Y)+c(Y,X)
 End
```

The updated BackPointers are now followed till another discrepancy is detected.

A motion planner has to find a path in some finite time or should report that no path is available. Resolution completeness of a planner guarantees to find a path if the resolution of the grid is fine enough. Most resolution complete planners are grid-based. The computational complexity of resolution complete planners is dependent on the number of points in the grid. It is given by O $(1/h^d)$, where h is the resolution (the length of one side of a grid cell) and d is the configuration space dimension.

III. Customization To D* Algorithm

D* algorithm has Open List and Closed List data structures to keep track of the explored and the unexplored cells. Closed List contains the explored cells while Open List contains the possible cells to which the rover can move from the current cell, which are not in the Closed List. Choosing the cell with the minimum cost is equivalent to finding the cell with the minimum cost in the Open List.

As Rover can move to only one of its adjacent cells, in this paper, Open List and Closed List are not made use of. Instead the cell with the minimum cost among the adjacent cells is chosen as the next cell. Further on identifying an obstacle, cost computation is redone not only for the current cell but also for those cells, which have the obstacle as its backPointer.

Consider the BackPointers shown in the Fig. 1. If current cell is the middle one and new obstacles are identified then cost updation takes place.

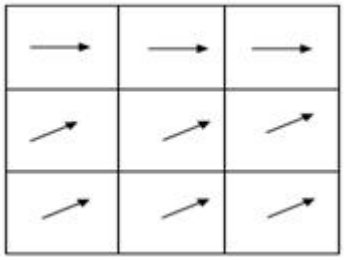

Figure 1: BackPointers showing the direction in which the rover must move to reach the target

Old cost functions are given in Fig. 2. Obstacles are represented with cells colored in red. Newly computed cost functions are given in Fig. 3.

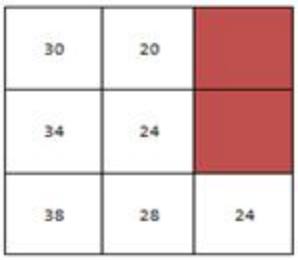

Figure 2: Old Cost function

Figure 3: Cost function Updated on identifying obstacles

On updating cost function for the current cell and its

DOI: 03.LSCS.2013.2.23

backPointers, the cell with minimum cost will be the one which is above the current cell with cost function value of 20. Then on moving to above cell, rover has to back trace its path as there is no path.

But when cost re-computation is done for cells which have obstacles as its backPointers, optimal path is chosen. Cell with minimum cost turns out to be the one with a value of 30. Rover can move to this cell and proceed towards the target point. Fig. 4 shows the new cost functions after cost updation.

Figure 4: Cost function updated for cells having obstacles in its backPointer list

IV. Simulation

D* was implemented in C++ and implementation of the logic was verified by testing under different scenarios.

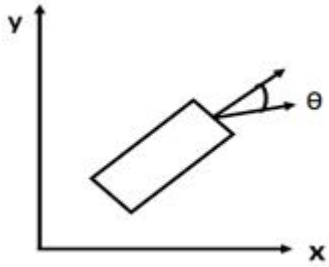

Figure 5: States of Rover

For the purpose of simulation an ideal rover has been considered where there is no wheel slippage. The rover has 3 states x, y, θ as shown in Fig.5. X, Y denotes the position of the rover in 2D, while θ gives the orientation.

It has been assumed that at any given instant, Rover can sense obstacles only in its adjacent eight cells. It records the sensed information for later use. Further, Rover could always move only to one of its adjacent cells.

As the terrain was logically split into squares the cost of traversing horizontally or vertically along the cell is the same. Moving diagonally involves more cost (1).

$$Cost(x,y)_{hor|ver} = p \text{ units}$$

$$Cost(x,y)_{diag} = \sqrt{2} * p \text{ units} \quad (1)$$

A 25 X 25 grid is considered where rows and columns range from 0 to 24. The starting point of the Rover is assumed to be (24, 0) while the target commanded is (0, 24). The shortest path traced has to be along the diagonal provided there are no obstacles.

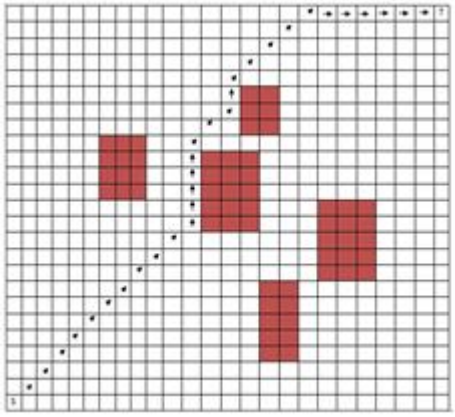

Figure 6: Path traced in scenario 1

Fig. 6 shows the path traced by the rover in the presence of obstacles. Cells which have Obstacles are filled in red color.

D* algorithm is efficient in solving dynamic environments. Fig. 7 illustrates the path traced by the rover in a different scenario. Fig. 8 illustrates another scenario successfully solved by D* algorithm.

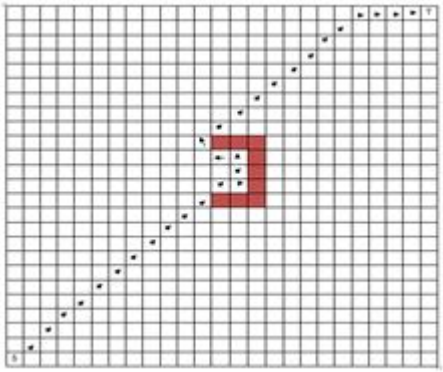

Figure 7: Path traced in scenario 2

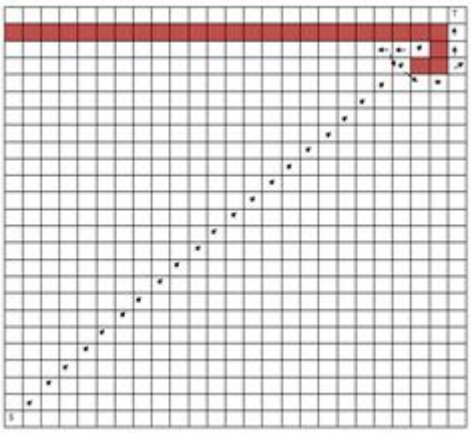

Figure 8: Path traced in scenario 3

V. Implementation In Lego Mindstorms Kit

D* was implemented in LEGO Mindstorms kit. The kit features a 32-bit ARM7 microcontroller with 256K flash and 64K RAM memory—running at 48MHz—and a second 8-bit AVR microcontroller with 4K flash and 512B RAM memory, running at 4MHz. ARM7 based processor runs the user

DOI: 03.LSCS.2013.2.23

program while the AVR processor controls the servos and sensors. The kit can be powered by a rechargeable battery pack.

Four input ports are available for connecting the sensors such as Touch sensor, Sound sensor, Light/color sensor, Ultrasonic sensor. Further, three output ports are also available for attaching the motors. The kit also contains three servo motors which have built in rotation sensors whose speed or position is controlled by a closed loop feedback circuit. Servos are controlled by an inbuilt PID algorithm. Servos can be commanded to operate at varying speeds. Rotation of two motors can also be synchronized.

For the purpose of experimentation Ultrasonic sensors have been used to identify the obstacles based on distance measurement. These sensors can measure distance from 3cm to more than 200cm. These Sensors have been mounted over a Servo to enable it to rotate to 360 degree, to sense obstacles on all sides.

RobotC, C like language, developed by Carnegie Mellon University was used to implement the D* logic in the Bot. The object code generated on compilation can be downloaded to the microcontroller. RobotC also provides a firmware (which can be thought of as the operating system of the NXT) which enables commanding the servos and sensors through high level programming language. Firmware is stored in Flash memory and thus it is non-volatile. Commands to servos can be controlled using power level to be applied to motor.

RobotC, being cross-platform, not only supports NXT but also RCX, Robotic Command eXplorers. It also provides better performance than NXT-G, due to the use of a particular RobotC specific firmware.

Fig. 9 shows the LEGO Bot with sensors. From the Ultra Sonic sensor readings, Bot updates the map of the region, with Walkable or Un-Walkable states. Then applying the D* algorithm, next cell to which Bot must move is identified. The process of sensing and moving continues till the target is reached.

Figure 9: LEGO Bot mounted with UltraSonic sensors

The pictorial view of readings given by Ultrasonic sensor when in front of an obstacle is shown in Fig. 10. The centre cell denotes the current location of the Bot. Shaded area of the circle is free of obstacles.

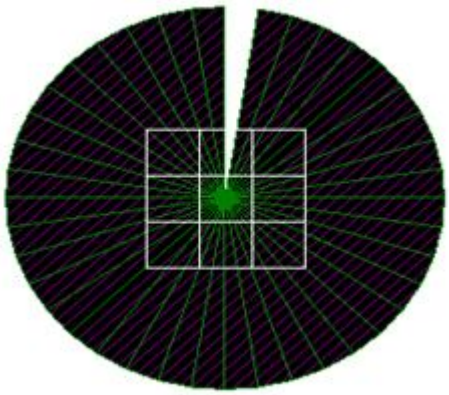

Figure 10: Ultrasonic sensor measurements

A. Cell Size

Determination of appropriate cell size is required for efficient implementation of the algorithm. A large cell size may not be very beneficial as the path computed may not be always optimal. If the cell size is small, then path planning will be efficient, but the rover itself may occupy several cells. Then on deciding the next movement, group of cells which are devoid of obstacles are to be identified. For the purpose of demonstration each cell was considered to be 40x40cm.

B. Cost Computation

The cost of moving 4 cms has been assumed to be 1 unit. Hence for moving along (vertically / horizontally) a cell of 40cms, cost involved would be 10 units and the cost of moving diagonally would be 14 units.

C. Calibration

Distance travelled by Bot for one full rotation of the wheel was computed. By accumulating the distance travelled from the wheel encoder values, position of the Bot was estimated. Since wheel encoders may not be accurate, calibration is necessary. The offset involved in one full rotation of the wheel was measured and the difference was absorbed in position estimation. Similarly, Ultra Sonic sensor was also calibrated.

D. Challenges Involved

While turning by 45 degree due to wheel slippage the orientation may not be accurate. On traversing a considerable distance the Bot may lose its orientation with respect to the terrain. To turn the Bot by a specified degree, differential commands were given to the servos. In order to orient the Bot accurately, while commanding the servos, the amount of differential movement was monitored and re-computed online, if required.

VI. Results

The implementation of D* in LEGO Mindstorms was evaluated in real time with different scenarios. One such

DOI: 03.LSCS.2013.2.23

scenario is explained below. A 4x4 grid was considered where each cell occupied 40cmx40cm. Fig. 11 shows the map of the grid considered, with the X and Y coordinates, where shaded cells contain obstacles. The Bot has to start at the point 'S' and reach the target 'T'.

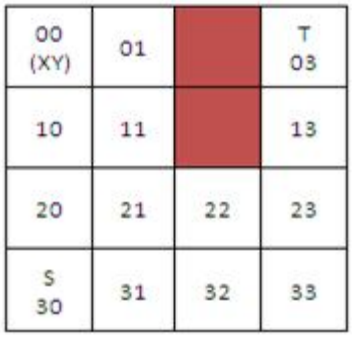

Figure 11: Grid considered

Initially all cells have been assumed to be walkable. Fig. 12 shows the backPointers stored for each cell to reach the target 'T', based on the assumption that all cells are traversable. As the Bot moves obstacles are identified and backPointers get updated accordingly.

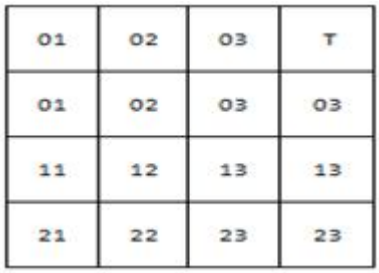

Figure 12: BackPointers of each cell

The path traversed by the Bot is shown in Fig. 13. At any instant it can sense only the adjacent 8 cells. Only on reaching cell – 11 the Bot could sense the obstacle in cell 02. So it has turned back and traced through the cells 22->13->03.

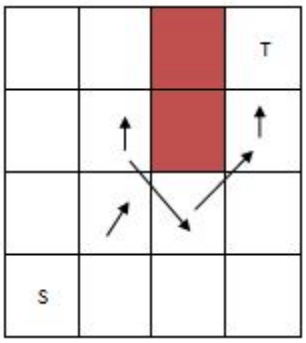

Figure 13: Path traversed

Modified backPointers for each cell on identifying the obstacles in cells 02 and 12 is given in Fig. 14.

D* is a promising path planning algorithm for unknown terrains. Further as the cost is recomputed in D*, dynamic cost variation can be implemented based on different terrains. For example the cost involved in climbing a slope can be assumed to be greater than moving over a plain surface. D* provides better solutions even when rover gets into deadlock

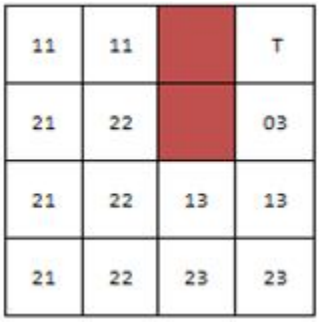

Figure 14: Modified BackPointers

situations in dynamically changing environments.

VII. Conclusion

The D* is a potential path planning algorithm for autonomous navigators. It is simulated in C++ and tested with various test cases. It is also implemented with NXT LEGO Mindstorms Bot using Ultasonic sensors and tested in real time environment. When the Bot was commanded to rotate, due to wheel slippage position and orientation of the Bot could not be accurately assessed. Errors have been accumulated as the Bot travelled some considerable distance. As future work, additional sensors such as Gyroscope are to be mounted in the Bot to accurately measure the rotation.

Acknowledgment

The authors wish to express their gratitude to Shri. Sudin Dinesh and to Shri. Arjun Narayanan, for their support in configuring the NXT LEGO Mindstorms kit.

References

[1] Steven M.LaValle, *Planning Algorithms*, Cambridge University Press 29/5/2006.

[2] GopiKrishnan S, BVS Shravan, Harishit Gole, Pratik Barve and Ravikumar L, "Path Planning Algorithms: A comparative study", *STS-2011*, VSSC, Trivandrum.

[3] Y. Koren, Senior Member, IEEE and J. Borenstein, Member, IEEE,"Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation", *Proceedings of the IEEE Conference on Robotics and Automation*, Sacramento, California, April 7-12, 1991, pp. 1398-1404.

[4] Jorge Nieto, Emanuel Slawinski, Vicente Mut and Bernardo Wagner, "Online Path Planning based on Rapidly-Exploring Random Trees", *IEEE International Conference on Industrial Technology (ICIT)*, 14-17, March 2010, Chile.

[5] Sven Koenig, Maxim Likhachev, David Furcy. "Lifelong Planning A*", *Artificial Intelligence Volume 155, Issues 1-2, Pages 93-146*, May 2004.

[6] Anthony Stentz, "Optimal and Efficient Path Planning for partially known environment", *IEEE International Conference on Robotics and Automation*, May 1994.

[7] Sobers L X Francis, Sreenatha G Anavatti, Matthew Garratt, "D* lite Search algorithm with Fibonacci heap for efficient Path Planning", *Proceedings of Advances in Control and Optimization of Dynamic Systems ACODS-2012*, February 2012, IISc Bangalore.

[8] Shaoyang Dong, Hehua Ju and Hongxia Xu, "An Improvement of D* lite Algorithm for Planetary Rover Mission Planning",

DOI: 03.LSCS.2013.2.23

Proceedings of the 2011 IEEE, International Conference on Mechatronics and Automation, August 7 - 10, Beijing, China

[9] Soh Chin Yun, Velappa Ganapathy, Tee Wee Chien, "Enhanced D* Lite Algorithm for Mobile Robot Navigation", 2010 IEEE Symposium on Industrial Electronics and Applications (ISIEA 2010), October 3-5, 2010, Penang, Malaysia.

[10] Sven Koenig, Maxim Likhachev, "Fast Replanning for Navigation in Unknown Terrain", IEEE Transactions on Robotics, Vol. 21, No 3, June 2005.

[11] Dave Ferguson and Anthony Stentz, "The Delayed D* Algorithm for Efficient Path Replanning", Proceedings of the 2005 IEEE International Conference on Robotics and Automation, April 18-22, 2005, Barcelona, Spain

[12] Xia Chen, Dong Liu, Ting Tang, "Path Planning for UCAV in Dynamic and Uncertain Environments Based on Focused D* Algorithm", 2011 Fourth International Symposium on Computational Intelligence and Design, 28 Oct - 30 Oct 2011, China

[13] Joseph Carsten and Arturo Rankin, Dave Ferguson, Anthony Stentz, "Global Planning on the Mars Exploration Rovers : Software Integration and Surface Testing", Journal of Field Robotics 26(4), 337–357, 2009.

DOI: 03.LSCS.2013.2.23